Itanium Virtualization and vNUMA

Matthew Chapman
University of NSW, National ICT Australia, HP Labs
matthewc@cse.unsw.edu.au

VIRTUALIZATION

What is virtualization?

- ➜ Normally, the operating system is part of the most privileged layer of system software
- ➜ Virtualization inserts another layer of software underneath the OS (usually transparently to the OS)

Scenarios

- ➜ Multiple OS instances on a single server
- ➜ Isolation guarantees between OS instances
- ➜ Migration of OS instances
- ➜ *etc.*

ITANIUM VIRTUALIZATION

Pure virtualization

- ➜ We demote the OS from privilege level 0 (most privileged) to 1
- ➜ The privileged instructions in the OS will now trap, and can be emulated with respect to a virtual machine
- ➜ Which instructions behave differently, and do not trap?
  - ➜ Called sensitive instructions

Itanium Instruction Set

Fixed point arithmetic

add	extr	pcmp	pshr	sxt
addp4	mix	pmax	pshradd	tbit
addl	movl	pmin	psub	tnat
and	mux	pmpy	shl	unpack
andcm	or	pmpyshr	shladd	xchg
cmp	pack	popcnt	shladdp4	xor
cmp4	padd	psad	shr	zxt
czx	pavg	pshl	shrp	
dep	pavgsub	pshladd	sub	

Floating point arithmetic

fabs	fcvt	fnma	fpms	fsetc
fadd	fma	fnmpy	fpneg	fsub
famax	fmax	fnorm	fpnegabs	fswap
famin	fmerge	for	fpnma	fsxt
fand	fmin	fpabs	fpnmpy	fxor
fandcm	fmix	fpack	fprcpa	getf
fchkf	fmpy	fpamax	fprsqrta	mov fr
fclass	fms	fpmerge	frcpa	setf
fclrf	fneg	fpmin	frsqrta	xma
fcmp	fnegabs	fpmpy	fselect	xmpy

Branch

br	brl	brp	chk	mov br

Register Stack Engine

alloc	flushrs	mov ar.bspstore
clrrrb	invala	mov ar.rsc [b]
cover [a]	loadrs	

[a] side-effect at privilege level 0 when psr.ic=0

[b] ar.rsc contains real privilege level

Memory access

cmpxchg	fwb	ldfp	probe	sync
fc [a]	ld	lfetch	st	
fetchadd	ldf	mf	stf	

Other unprivileged

break	mov ar	mov ip	nop	srlz
epc	mov cpuid [b]	mov pr	rum	
hint		mov um	sum	

[a] bypasses protection check at privilege level 0

[b] returns cpuid of real processor

Memory management

itc	mov pkr	ptc	tak	tpa
itr	mov rr	ptr	thash [a]	ttag [a]

Other privileged

bsw	mov dbr	mov pmc	mov psr	rsm
mov cr	mov ibr	mov pmd [b]	rfi	ssm

[a] not privileged

[b] unprivileged reads return 0 instead of trapping

OTHER ISSUES

Fixed set of privilege levels

- ➜ Guest PL 0..3 must be mapped onto 1..3
- ➜ Possible loss of protection

Exception handlers live in virtual memory

- ➜ Need to co-ordinate address with guest kernel

Complex translation modes

- ➜ Separate control over instruction/data/RSE translation
  e.g. data physical, register stack virtual
- ➜ Difficult to emulate in virtual mode

Register Stack Engine

- ➜ Not completely virtualizable
- ➜ Partially loaded frames cannot exist outside CPL0

Dealing with Non-virtualizable Architectures

Static translation (e.g. vBlades, vNUMA)

➜ Preprocess OS binary substituting sensitive instructions

Dynamic translation (e.g. VMware)

➜ Scan and translate sensitive instructions at runtime

Para-virtualization (e.g. Xen, VMI)

➜ Manually modify guest operating system

Hardware support (e.g. Intel VT-i)

➜ Modify the architecture to close virtualization holes

INTEL VIRTUALIZATION TECHNOLOGY FOR ITANIUM (VT-I)

First introduced in Montecito processor (aka Dual-Core Itanium 2)

- ➜ New `psr.vm` control bit
- ➜ All sensitive instructions fault with `psr.vm=1` (even in CPL0)
- ➜ One bit of address space reserved for hypervisor
  (allows guest kernel and hypervisor to safely cohabit CPL0)
- ➜ `vmsw` instruction for entering/exiting hypervisor (analogous to `epc`)

BUT...

Pure virtualization is expensive!

- ➜ Trap for every privileged instruction
- ➜ Need to read instruction from memory, decode and emulate
- ➜ Some privileged instructions are very common
  (e.g. enabling/disabling interrupts)

CUTTING THE COST OF VIRTUALIZATION

Virtualization acceleration

- ➜ VT-i provides a framework that could support acceleration hardware in the processor
- ➜ However, roadmap for this hardware is not clear
- ➜ Multicore design trends may pressure architects to limit core complexity rather than increasing it

Cutting the Cost of Virtualization

Para-virtualization (e.g. Xen, VMI)

- ➜ Custom interface to hypervisor
- ➜ Guest kernel needs to be ported to this interface
- ✓ Good performance possible
- ✗ Porting is time-consuming and error-prone

Optimised para-virtualization (e.g. vBlades)

- ➜ Only para-virtualize the performance-critical parts
- ✓ Good compromise, time-performance tradeoff can be chosen
- ✗ Still involves manual porting

Pre-virtualization (automated para-virtualization)

- ➜ Privileged instructions automatically substituted at assembly time

PRE-VIRTUALIZATION

Step 1. Preprocessor transforms instructions to allow macro replacement:

```
(p6) mov r16=cr.iip
```

```
emul_read_cr pr=p6,dst=r16,cr=19
```

PRE-VIRTUALIZATION

Step 2. Macro definition file implements macros for a particular hypervisor:

```
.macro emul_read_cr pr=p0,dst,cr
(\pr)   mov \dst=__vnuma_cpu+CPU_CR_OFFSET+8*\cr
        ;;
(\pr)   ld8 \dst=[\dst]
.endm
```

PRE-VIRTUALIZATION

Step 2 (alternative). Macro definition file just adds nop padding, hypervisor rewrites binary at load time.

```
.macro emul_read_cr pr=p0,dst,cr
9998:
(\pr)   mov \dst=cr\cr
(\pr)   nop.m 0x0
9999:
.pushsection .afterburn
.quad 9998b
.quad 9999b
.popsection
.endm
```

LMBENCH RESULTS

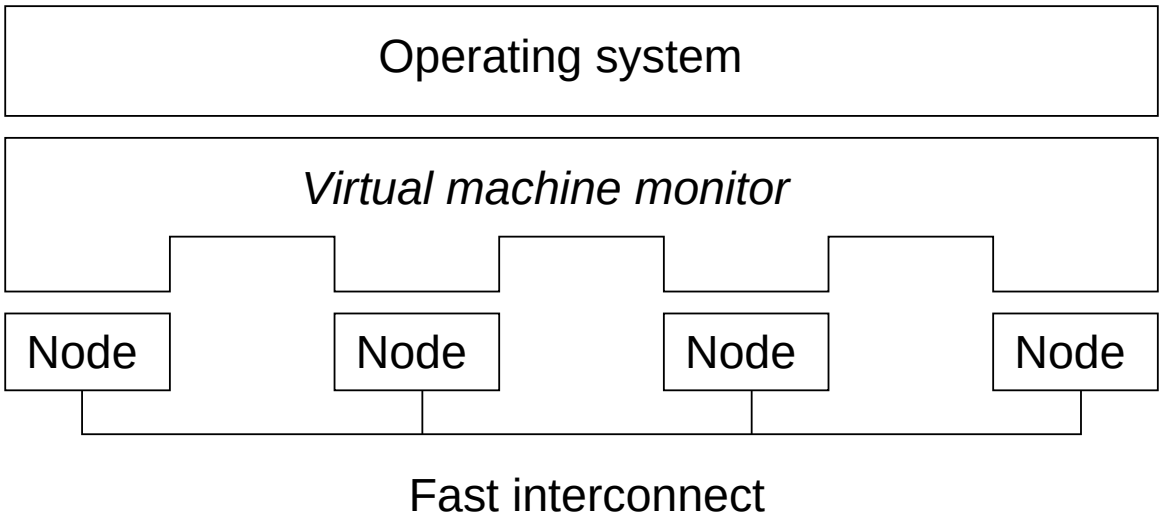
Basic latencies (Xen, in μs)

Benchmark	Native	Pure	Para	Pre
null call	0.04	0.96	0.50	0.04
null I/O	0.27	6.32	2.91	0.42
stat	1.10	10.69	4.14	1.43
open/close	1.99	20.43	7.71	2.60
install sighandler	0.33	7.34	2.89	0.50
handle signal	1.69	19.26	2.36	2.23
fork	56	513	164	152
exec	316	2084	578	566
fork+exec sh	1451	7790	2360	2231

vNUMA

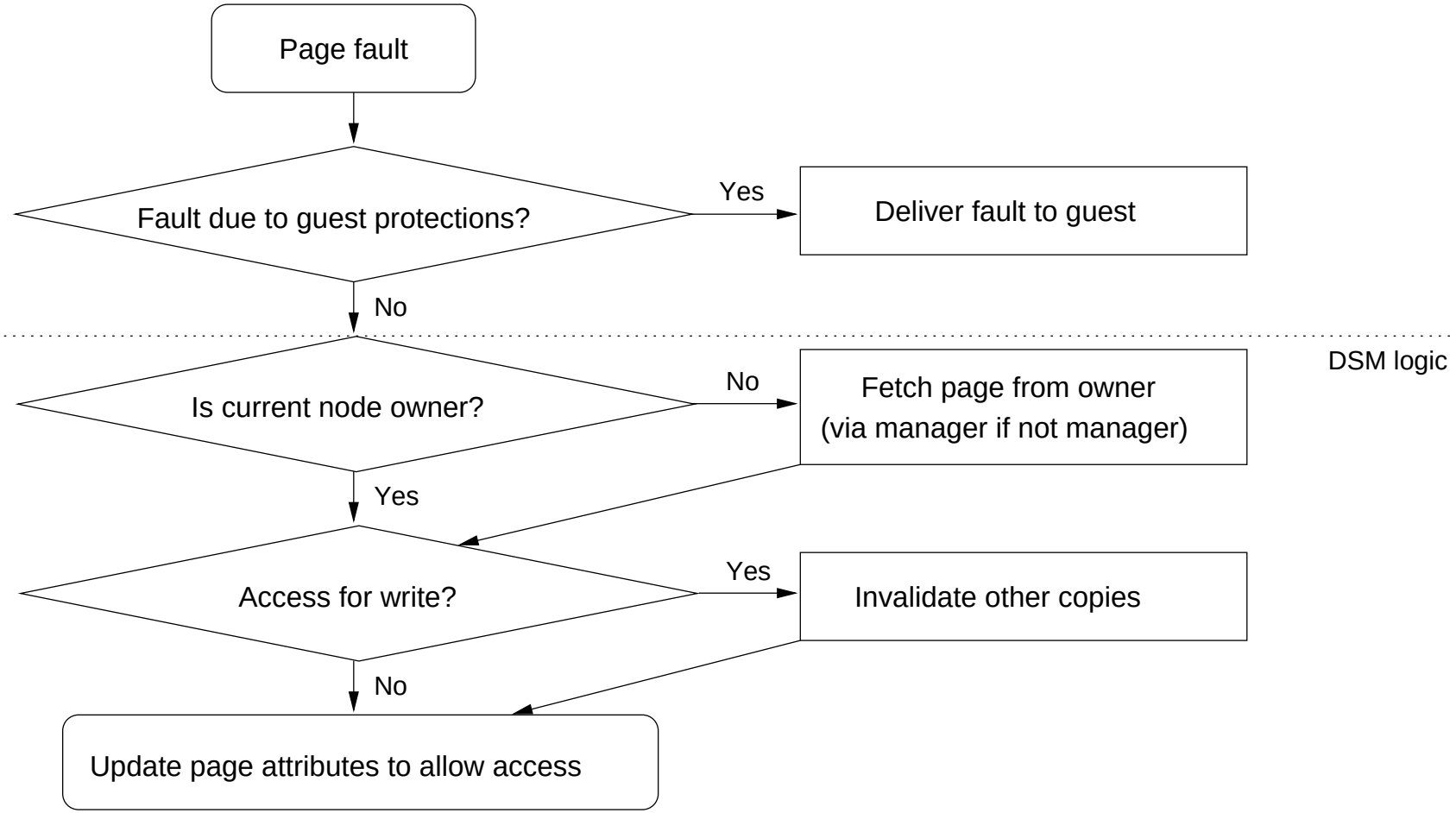
- Itanium virtual machine monitor developed at UNSW
- Thin hypervisor, no host OS
- Optimised for performance
  - Written mostly in C, but non-standard runtime conventions to minimise cost of entry to C
  - Runs with `psr.ic=0`, no nested exceptions allowed
- Supports pre-virtualized guests for better performance
- Distributed!

vNUMA Distribution



- ➜ Start vNUMA on multiple nodes of a cluster
- ➜ vNUMA locates and manages CPU/memory/IO resources
- ➜ Presents illusion of a single large NUMA machine to guest OS
- ➜ Memory is globally visible using paging techniques (distributed shared memory)

Dirty/Accessed bits are overloaded for DSM use



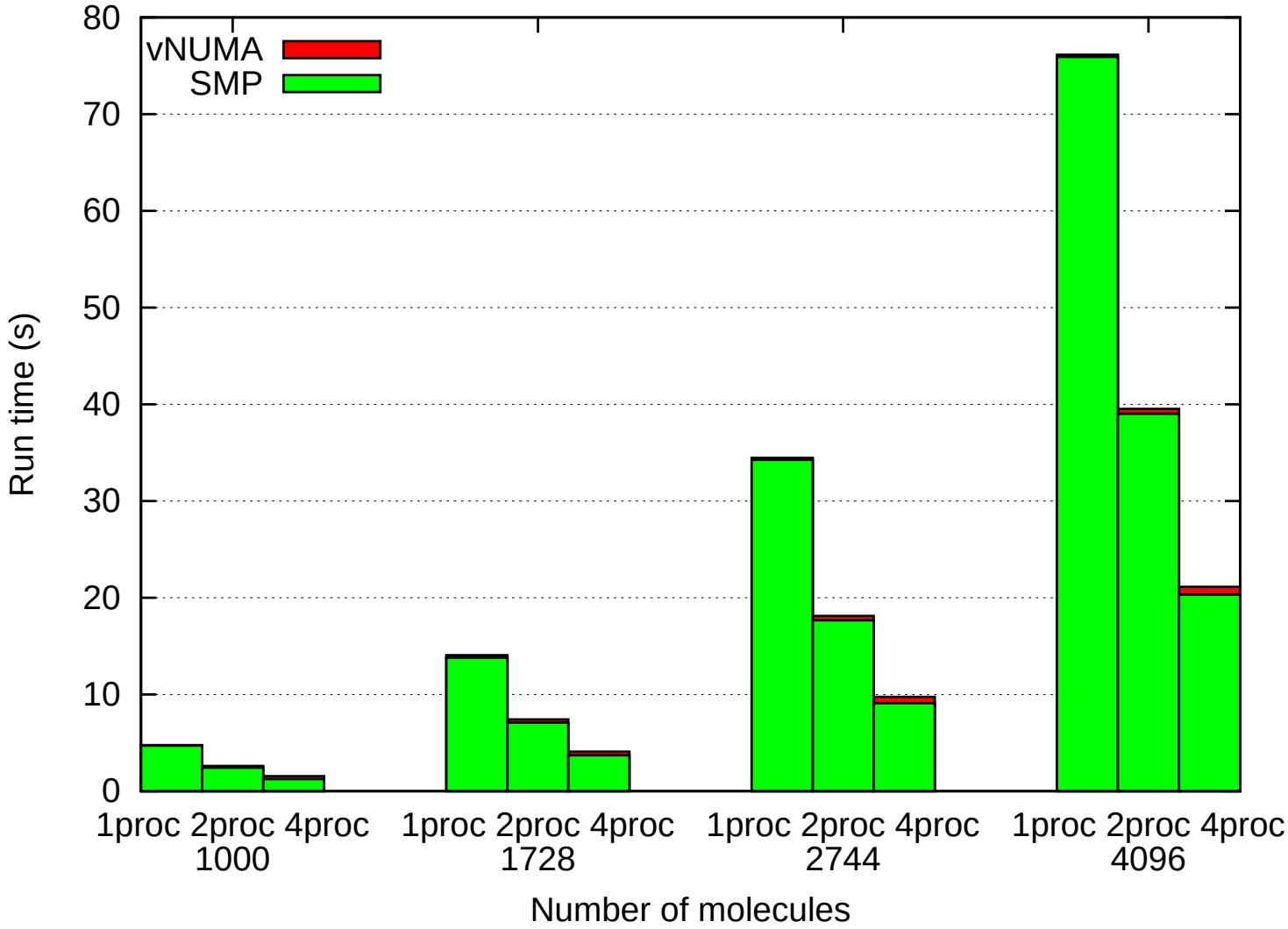
EVALUATION

Stanford Parallel Applications for Shared Memory (SPLASH-2)

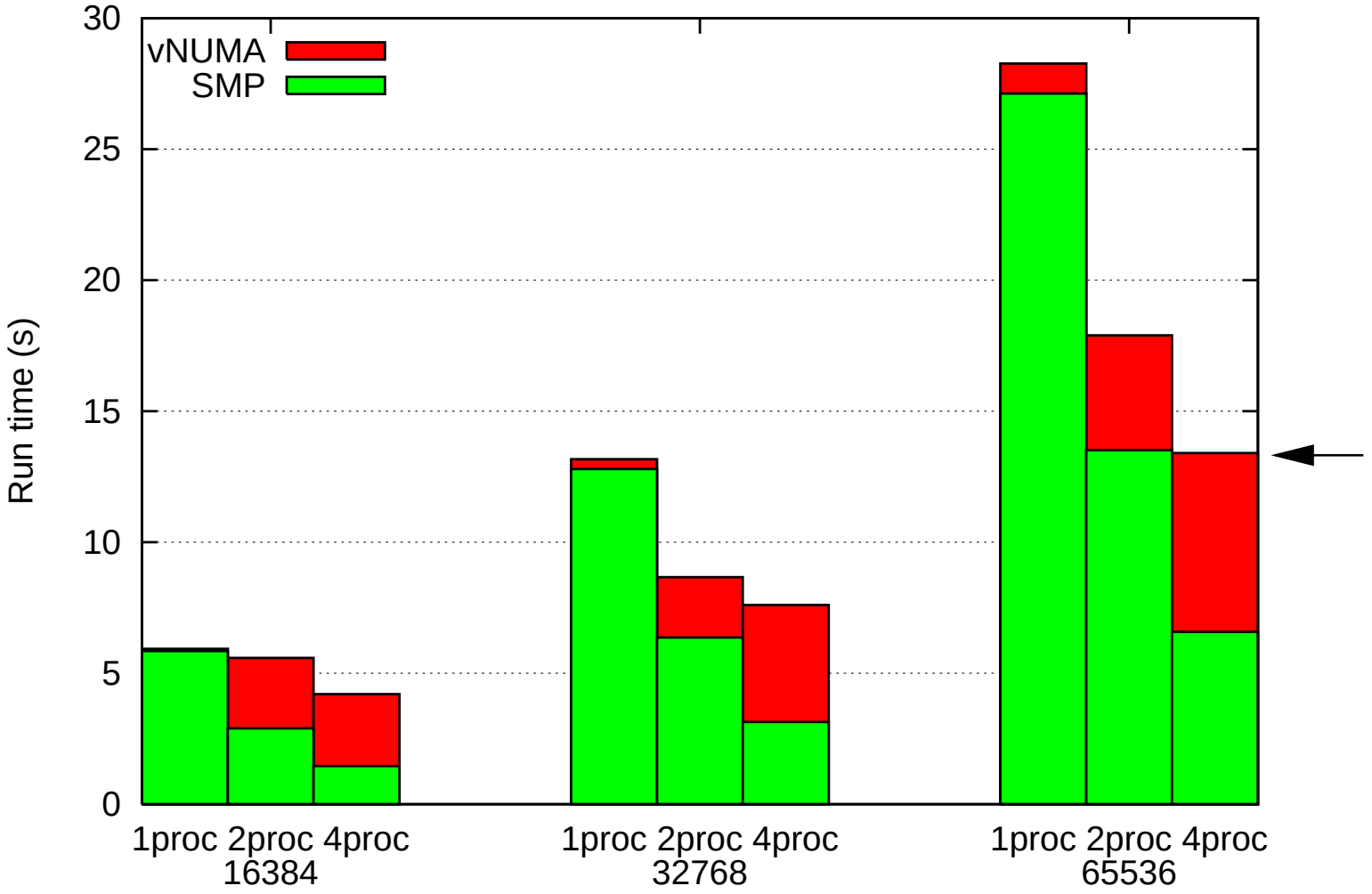
- ➜ Set of scientific applications and kernels
- ➜ Compare N-node vNUMA cluster to N-way SMP

WATER-NSQUARED

➜ Known to perform well on a DSM system

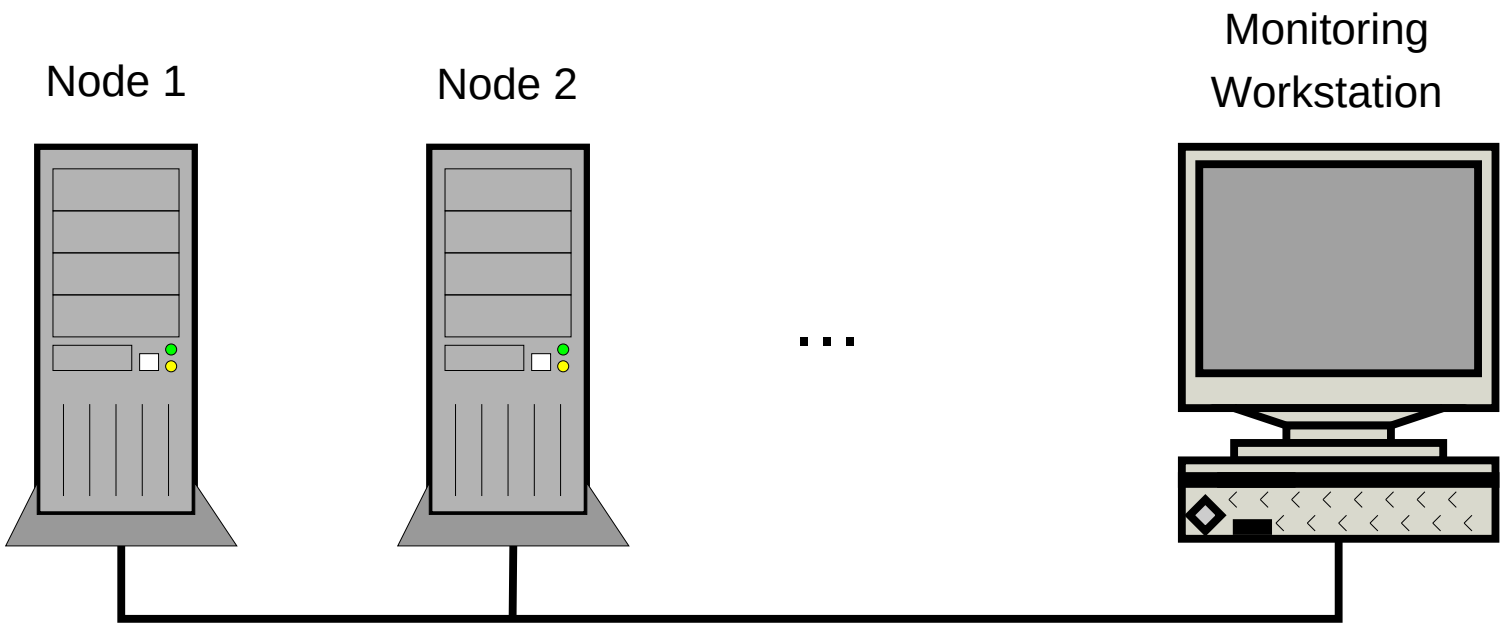


BARNES



➜ Some issues with lock contention and sparse writes

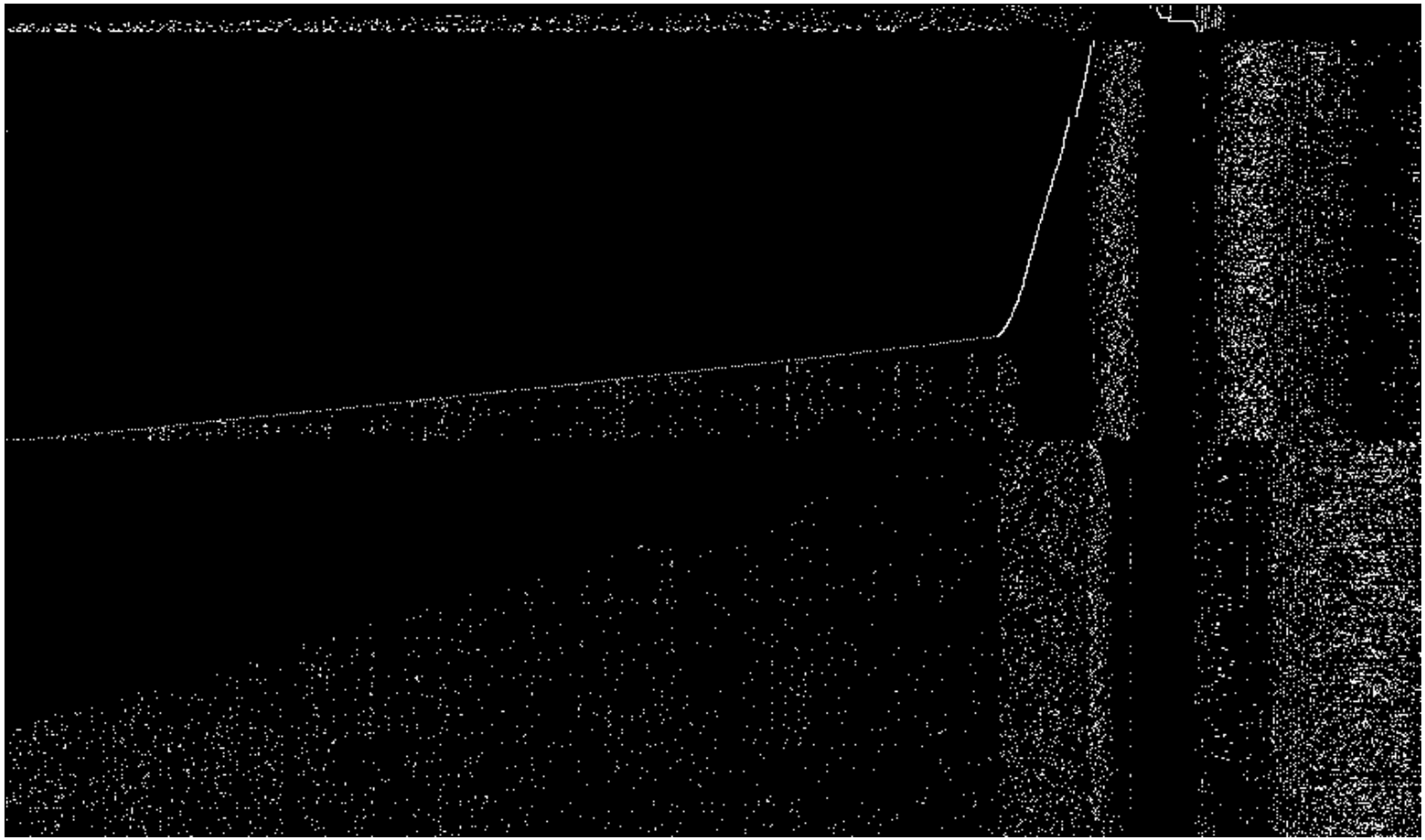
Profiling Methodology



- Each node collects profiling data in a buffer
- Node regularly sends buffer to monitoring workstation
- Monitoring tools: `vnumaviz`, `vnumaprof`

VNUMAVIZ

➜ Space-time plot of faults



Time

VNUMAPROF

➜ Shows the instructions that resulted in the most DSM activity

```
Total DSM overhead: 102521.875 ms in 138924r/45x/79600w/61623i
             User: 97876.873 ms in 132946r/29x/76453w/57661i
           Kernel: 4645.002 ms in 5978r/16x/3147w/3962i
0x400000000000aa20: 18744.359 ms in 49346r/0x/0w/0i
 in gravsub() at /home/matthewc/splash2/codes/apps/barnes/grav.C:71
         SUBV(Local[ProcessId].dr, Pos(p), Local[ProcessId].pos0);
0x40000000000042a0: 17781.555 ms in 31380r/0x/0w/0i
 in stepsystem() at /home/matthewc/splash2/codes/apps/barnes/code.C:578
        MULVS(dvel, Acc(p), dthf);
0x40000000000044a0: 11078.159 ms in 0r/0x/17930w/17882i
 in stepsystem() at /home/matthewc/splash2/codes/apps/barnes/code.C:581
        ADDV(Pos(p), Pos(p), dpos);
0x40000000000004d40: 10260.499 ms in 0r/0x/27081w/27433i
 in ComputeForces() at /home/matthewc/splash2/codes/apps/barnes/code.C:643
       Cost(p)=0;
```

KERNEL COMPILE

Linux kernel compile benchmark

- ➜ Parallel make (`make -j4`)
- ➜ Still shows negative speed-up
- ➜ Currently under investigation

Linux Kernel Compile Take 1 - SMP

```
Total DSM overhead: 230514.701 ms in 209433r/12x/252838w/105845i
              User: 38112.312 ms in 59456r/12x/31020w/18541i
            Kernel: 192402.389 ms in 149977r/0x/221818w/87304i
0xa0000001002a1df0: 28793.682 ms in 0r/0x/66726w/7121i
 in clear_page()
0xa00000010045af40: 10722.957 ms in 0r/0x/18059w/10602i
 in __raw_spin_lock_flags() at include/asm/spinlock.h:78
0xa0000001002a1da0: 8861.123 ms in 0r/0x/21813w/2890i
 in clear_page()
0x400000000003bf20: 8372.198 ms in 22071r/0x/0w/0i
0xa0000001000d4430: 4297.463 ms in 10209r/0x/0w/0i
 in get_page_from_freelist() at mm/page_alloc.c:816
0xa0000001000d4190: 3692.115 ms in 0r/0x/6989w/6460i
 in __list_del() at include/linux/list.h:151
...
```

Linux Kernel Compile Take 2 - NUMA-aware

```
Total DSM overhead: 274433.858 ms in 338629r/376x/243140w/168895i
              User: 16178.721 ms in 28739r/373x/15045w/11125i
            Kernel: 258255.137 ms in 309890r/3x/228095w/157770i
0xa000000100014fb0: 36058.775 ms in 72762r/0x/0w/0i
 in default_idle() at arch/ia64/kernel/process.c:205
0xa000000100465ce0: 22719.967 ms in 0r/0x/42908w/39275i
 in __raw_spin_lock_flags() at include/asm/spinlock.h:78
0xa0000001002ad790: 6576.274 ms in 0r/0x/18188w/2116i
 in clear_page()
0xa0000001000663a0: 5142.599 ms in 0r/0x/4837w/5281i
 in __raw_spin_unlock() at include/asm/spinlock.h:94
0xa00000010006e960: 4535.700 ms in 11647r/0x/0w/0i
 in source_load() at kernel/sched.c:950
...
```

PLANS

Current directions

- Improvements based on profile feedback
- Special handling of spinlocks?
- I/O improvements
- Multithreading to mask network latency
  - Run other virtual CPUs while waiting for a page
- Public release